Java Exception Types: Built-In And Custom

What Is an Exception In Java?

Imagine you're a student preparing for an important exam. But on the exam day, you fall sick and can't make it. In Java, exceptions are like unexpected disruptions that disrupt the normal flow of your program. They can occur due to bad input, unavailable resources, or unexpected errors. Just as you need a backup plan when things go wrong, handling exceptions in Java allows your program to respond to problems gracefully.

Now to handle the exception, we must know what type of exception.

Types Of Java Exception

Exceptions in Java can be categorized into two ways i.e.

1. Built-In Exceptions.
   - Checked Exceptions.
   - Unchecked Exceptions.
2. Customized Exceptions.

Built-In Exceptions

Built-in exceptions, also known as standard exceptions, are predefined exception classes provided by Java. These exceptions cover a wide range of common errors and exceptional situations that can occur during program execution. Built-in exceptions are part of the Java standard library and provide a standardized way to handle common exceptional scenarios.

Built-In Exceptions can be further classified into two categories –

1. Checked Exceptions
2. Unchecked Exceptions

Checked Exceptions

Checked exceptions are exceptions that the Java compiler requires you to handle explicitly in your code. These exceptions are checked at compile-time, and if not handled properly, the code will not compile. Checked exceptions are subclasses of the Exception class but not subclasses of the RuntimeException class.

Here are some key points about checked exceptions:

- The Java compiler ensures that all checked exceptions are either caught or declared to be thrown.
- Checked exceptions are used for scenarios where it's possible to recover from the exception, and the programmer is forced to handle the exception explicitly.
- Some examples of checked exceptions in Java include IOException, SQLException, and ClassNotFoundException.

Example 1: FileNotFoundException Exception

In Java, the `FileNotFoundException` is a type of exception that occurs when a file is requested but cannot be found or accessed. It is a subclass of the `IOException` class and is commonly encountered when working with file operations.

Code:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;


public class FileReadExample {
```

```
    public static void main(String[] args) {
        try {
            File file = new File("example.txt");
            Scanner scanner = new Scanner(file);
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                System.out.println(line);
            }
            scanner.close();
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        }
    }
}
```

Output:

```
File not found: example.txt (No such file or directory)
```

In this example, we're attempting to read the contents of a file named `example.txt`. However, if the file does not exist in the specified location, a `FileNotFoundException` will be thrown. The code uses a `try-catch` block to handle this exception.

Example 2: ClassNotFound Exception

In Java, the `ClassNotFoundException` is a type of exception that occurs when a class is not found at runtime. It typically happens when the Java Virtual Machine (JVM) tries to load a class dynamically using methods like `Class.forName()` or `ClassLoader.loadClass()`, but the specified class cannot be found in the classpath.

Code:

```
public class ClassNotFoundExample {
    public static void main(String[] args) {
        try {
            Class clazz = Class.forName("com.example.NonExistentClass");
        } catch (ClassNotFoundException e) {
            System.out.println("Class not found: " + e.getMessage());
        }
    }
}
```

Output:

```
Class not found: com.example.NonExistentClass
```

In this example, we're trying to load a class named `com.example.NonExistentClass` dynamically using the `Class.forName()` method. However, if the specified class does not exist in the classpath, a `ClassNotFoundException` will be thrown. The code uses a try-catch block to handle this exception.

Example 3: IllegalAcess Exception

In Java, the `IllegalAccessException` is a type of exception that occurs when a program tries to access a field, method, or constructor of a class that is not accessible due to access modifiers such as `private`, `protected`, or `package-private`.

Code:

```
public class IllegalAccessExample {
    private int privateField;

    public static void main(String[] args) {
        IllegalAccessExample example = new IllegalAccessExample();

        try {
            Class clazz = example.getClass();
            java.lang.reflect.Field field = clazz.getDeclaredField("privateField");
            field.setAccessible(true);
            int value = (int) field.get(example);
            System.out.println("Private field value: " + value);
        } catch (NoSuchFieldException | IllegalAccessException e) {
            System.out.println("Illegal access: " + e.getMessage());
        }
    }
}
```

Output:

```
Illegal access: Attempted to access a private field
```

In this example, we have a class `IllegalAccessExample` with a `private` field named `privateField`. We try to access this private field using Java Reflection by setting its accessibility to true using `field.setAccessible(true)`. However, if the access modifier of the field prevents its access, an `IllegalAccessException` will be thrown.

Unchecked Exceptions

Unchecked exceptions, also known as runtime exceptions, are exceptions that do not need to be explicitly handled in the code. These exceptions are subclasses of the

RuntimeException class or its subclasses. Unlike checked exceptions, unchecked exceptions are not required to be declared or caught explicitly.

Here are some key points about unchecked exceptions:

- Unchecked exceptions are checked at runtime rather than at compile time.
- Unchecked exceptions are typically used for scenarios that are not recoverable or unexpected, such as NullPointerException or ArrayIndexOutOfBoundsException.
- Some examples of unchecked exceptions in Java include NullPointerException, ArrayIndexOutOfBoundsException, and IllegalArgumentException.

Example 1: Arithmetic Exception

In Java, the `ArithmeticException` is a type of exception that is thrown when an arithmetic operation encounters an error or an exceptional condition. It typically occurs when performing arithmetic operations such as division by `zero` or an `overflow/underflow `condition.

Code:

```
public class ArithmeticExceptionExample {
    public static void main(String[] args) {
        int dividend = 10;
        int divisor = 0;

        try {
            int result = dividend/divisor;
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Output:

```
Error: / by zero
```

In this example, we have an arithmetic operation where we attempt to divide `dividend` by `divisor`. However, the value of the divisor is set to zero, which results in an arithmetic error. The division by zero operation throws an `ArithmeticException`.

Example 2: ArrayIndexOutOfIndex Exception

In Java, the `ArrayIndexOutOfBoundsException` is a type of exception that is thrown when attempting to access an array element in array of Java using an invalid index. It occurs when the index used to access an array is either negative or greater than or equal to the length of the array.

Code:

```
public class ArrayIndexOutOfBoundsExceptionExample {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3};

        try {
            int element = numbers[3];
            System.out.println("Element: " + element);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Output:

```
Error: Index 3 out of bounds for length 3
```

In this example, we have an array of numbers with three elements. However, we attempt to access the element at index 3, which is outside the valid index range of the array. This leads to an `ArrayIndexOutOfBoundsException` being thrown.

Example 3: NullPointer Exception

In Java, the `NullPointerException` is a type of exception that is thrown when trying to access or perform operations on a null object reference. It occurs when an object reference variable doesn't point to any valid object in memory but is used as if it does.

Code:

```
public class NullPointerExceptionExample {
    public static void main(String[] args) {
        String name = null;

        try {
            int length = name.length();
```

```
            System.out.println("Length: " + length);
        } catch (NullPointerException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Output:

```
Error: Cannot invoke "String.length()" because "name" is null
```

In this example, we have a `String` variable name initialized with a null value. We then try to access the `length()` method on this null reference, which leads to a `NullPointerException` being thrown.

Checked and Unchecked Exceptions are not limited to the above examples. These are just a few examples of checked and unchecked exceptions. You can refer to another blog to know more in detail.

Difference Between Checked Vs Unchecked Exception

Checked Exceptions	Unchecked Exceptions
Must be declared in the method signature or handled using try-catch blocks or throws clause.	Do not require explicit declaration or handling mechanisms.
Checked exceptions pertain to conditions that are expected to occur and can be handled gracefully.	Unchecked exceptions are typically caused by programming errors or exceptional situations.
The compiler enforces the handling or declaration of checked exceptions by	The compiler does not enforce the handling of the declaration of unchecked exceptions.

requiring try-catch blocks or the throws clause.	
Checked exceptions are part of the Java checked exception hierarchy.	Unchecked exceptions do not follow the checked exception hierarchy.
Checked exceptions must be explicitly caught or declared in the method signature.	Unchecked exceptions are not required to be explicitly caught or declared.
Examples of Checked exceptions:FileNotFound ExceptionNoSuchField ExceptionInterrupted ExceptionNoSuchMethod ExceptionClassNotFound ExceptionIO Exception	Examples of Unchecked Exceptions:NullPointer ExceptionNumberFormat ExceptionIllegalState ExceptionArrayIndexOutOfBounds ExceptionSecurity ExceptionArithmetic Exception